CS200: Programming I **Instructor:** Sam Vinitsky

P3: Marian the Librarian

(20 points)

Due: Tuesday, July 2 @ 11:59pm

Part 0: Introduction

Brrrring brriiiiiing the phone rings – it's Marian, the local librarian! She heard you were taking an introductory computer science course, and has decided to enlist your help with writing some computer programs to help her run the library. Marian is a fantastic librarian, but unfortunately very bad at math. She needs you to write a variety of functions to perform basic math tasks for her.

Marian's computers are slow. Luckily, they **can** run "if/else" statements! However, they cannot run any of the following: loops, String parsing, built-in Java libraries, or anything we havent explicitly covered in class.[1] **In particular, you may not use any functions from the built-in Java "Math" library.** If you have questions about what you are/not allowed to use, please ask on Piazza.

Before attempting this homework, work through all of the practice problems from this week's lecture (posted on Canvas here).

Each program you turn in should include a comment at the top with (1) your full name, (2) your student ID number, (3) your netID, and (4) the name of anyone you discussed the homework with (excluding Sam and Alex).

As always: **start early, ask questions, and have fun!**

Part 1: Teenage Wasteland [3 points]

Marian's library is facing a major epidemic. Some of the local schoolchildren have been checking out *adult* books that are entirely inappropriate for their age. These books contain topics and illustrations that might scare children – including instructions for building a bomb, nude paintings of women, and detailed descriptions of property tax laws.

[1] If you don't know what any of these are, then don't worry about it, because you likely won't accidentally use something we haven't learned about...

In light of this, Marian needs you to write a program to help figure out which books a given person is allowed to borrow. Write a program called `AgeRestriction.java`, and include the functions described in this section.

As usual, make sure to test that your functions are working correctly by putting various print statements in `main`!

a) Over 13? [1 point]

Marian has decided that kids under the age of 13 should **not** be allowed to borrow books, because they are too much of a liability. Unfortunately, Marian is **very** bad at math, so when someone tells her their age, she has trouble telling whether it is greater than or less than 13. This is where you come in!

Write a function called `isOver13` that takes an integer, and returns `true` if it is greater than or equal to 13, and `false` otherwise.

Function Name:	`isOver13`
Input(s):	`int age`: the borrower's age
Return value:	`boolean`: `true` if age is 13 or above, `false` otherwise
Special requirements:	None
Hints:	Use a conditional statement(s)...
Example usage:	`isOver13(5)` should return `false` `isOver13(20)` should return `true`

b) Old enough for taxes? [2 point]

If the borrower is 13 or older, but still under 18, they will not be allowed to borrow any books containing sensitive materials, but will still be allowed to borrow anything else. As before, Marian needs your help determining whether ages fall in that range or not.

Write a function called `whatCanIBorrow` that takes an integer, and prints out what they can borrow. This function will return **nothing**. See the table for details.

Function Name:	`whatCanIBorrow`
Input(s):	`int age`: the borrow's age
Return value:	Nothing!
Printout:	- If 18 or older, print: `You can borrow any book!!!` - If over 13, but under 18: `You cannot borrow sensitive material.` - If under 13: `You cannot borrow any books.` - If age is negative, or over 100: `INVALID AGE`
Hints:	Use `if` and `else` statements

Example usage:	`whatCanIBorrow(5)` should print out
	`You cannot borrow any books.`
	`whatCanIBorrow(15)` should print out
	`You may not borrow sensitive material.`

Part 2: Everything in its Place [3 points]

Because the teenagers were furious about the new policy limiting which books they could borrow, they *rampaged* through the library, throwing all the books on the floor. Marian needs your help putting all the books back in the correct order!

Each book is associated with an integer "ID number" that indicates where in her collection the book should go. Unfortunately, the hoodlums also *stole* some of the books, so she is missing some numbers![2] Marian has trouble with numbers (still), so she needs your help figuring out which order the books should be placed in.

Write a program called `OrganizeBookshelf.java`, and include the functions described in this section.

Make sure to test your functions!

a) Which book is right? (1 point)

Marian is picking up the books from the hap-hazard pile the rampant teens threw them in. She can only hold two books at once (one in each hand), and needs to figure out which of the two should be sorted on the **right** of the other.

Marian wants to sort her shelves so that the IDs increase from left the right – i.e. if the book IDs are 1,5,2,10, then the left-most book should be the book with ID 1, then 2, then 5, and the right-most book with have ID 10.

[2] Also, some of the ID numbers are negative, just for fun.

Write a function called `getRightmostBookID` that takes two `int`'s representing ID numbers, and returns which number will be placed further right on the shelf.

Function Name:	`getRightmostBookID`
Input(s):	`int id_1`: book 1's ID number `int id_2`: book 2's ID number
Return value:	`int`: ID of the book that should be placed on the right
Special requirements:	None
Hints:	They will never be the same...

Example usage:	`getRightmostBookID(5, 10)` should return `10`
	`getRightmostBookID(111, 10)` should return `111`
	`getRightmostBookID(-100, -5)` should return `-5`
	`getRightmostBookID(-100, 5)` should return `5`

b) Distance between books (2 point)

Marian is missing many books from her library that were stolen by the rampaging teenagers, but she hopes to get them back someday. When placing books on her shelf, she wants to leave room between them for all the books with IDs between them. That is, if she were to shelve the books with IDs 10 and 15, she would need to leave 4 spaces between them for books 11,12,13, and 14.

Write a function called `spaceToLeave` that takes two integers, and returns how much space Marian needs to leave between them on the shelves.

Remember, you may **not** use the `Math` module, including `Math.abs()` (we haven't even remotely gotten to this in class).

Function Name:	`spaceToLeave`
Input(s):	`int id_1`: book 1's ID number `int id_2` book 2's ID number
Return value:	`int` that represents the number of integers/spaces between `id_1` and `id_2`
Special requirements:	None
Hints:	Is this just the difference between them? Does the order of the inputs matter?

Example usage:	`spaceToLeave(5,15)` should return `9`
	`spaceToLeave(10,0)` should return `9`
	`spaceToLeave(-5,10)` should return `14`
	`spaceToLeave(5,-10)` should return `14`

Part 3: A Bad, Bad, Bad Account [3 points]

Because Marian's controversial policies on censorship have alienated a lot of the library's "customers", the library has had a major problem with people borrowing books without returning them, just to make her life harder! If someone has a book out more than a week past the date they were supposed to return it, they owe the library $10.

Write a program called `LateFees.java` that contains the functions described in this section.

Make sure to test your functions!

a) Whose account is bad? (2 points)

Due to an issue with the old system of keeping track of late fees, many people were made to pay the $10 late fee who did not actually owe the library money. There are many people who have accounts for the library, so it would be nice to write a function that determines which accounts need to be examined.

An account is considered **bad** if either (a) the account paid money, but has no overdue books, or (b) the account has not paid, but does have an overdue book.

Write a function called `isAccountBad` that takes two `boolean` variables as input representing whether they have an overdue book, and whether they have paid the late fee. This function should return `true` if **exactly** one of the inputs is `true`.

Function Name:	`isAccountBad`
Input(s):	`boolean paidLateFee`: has this account paid a late fee? `boolean hasOverdueBook`: does this account have an overdue book?
Return value:	`boolean` that is `true` if the account is (a) paid and not overdue, or (b) overdue and not paid.
Special requirements:	None
Hints:	This can be done with a single if/else statement
Example usage:	`isAccountBad(true, true)` should return `false` `isAccountBad(false, true)` should return `true`

b) Whose account is good? (1 points)

Marian also needs a way of keeping track of which accounts are **good**! Write a function called `isAccountGood` that return `true` if the account has settled all of its overdue fees, or never had any (and never paid any!).

Function Name:	`isAccountGood`
Input(s):	`boolean paidLateFee`: has this account paid a late fee? `boolean hasOverdueBook`: does this account have an overdue book?
Return value:	`boolean` that is `true` if the account is (a) paid and overdue, or (b) not overdue and not paid.
Special requirements:	None
Hints:	You can call functions you've written in previous parts... What does `!` do?
Example usage:	`isAccountGood(true, true)` should return `true` `isAccountGood(false, true)` should return `false`

Part 4: Do it for the children! [8 points]

In an attempt to foster goodwill with the community, Marian has started a weekly club for children to come and play educational "games" at the library. Their favorite game is about **combining numbers**. Since Marian is bad at math, she is bad at facilitating the game, so she needs your help!

Write a program called `Combinations.java` with the functions described below.

a) Simple Combinations (3 points)

The game they play goes as follows: Marian gives the kids three whole numbers, and they need to determine whether any two of them can be **added** or **multiplied** to produce the third (the order of the numbers doesn't matter).

For example, if the kids are given `10`, `50`, and `5`, the correct answer is "multiplied", since `10*5` is equal to `50`. If the kids are given `11`, `121`, `11`, again the answer is "multiplied", since `11*11` is `121`.

Marian has trouble telling when the kids are right or wrong! It's your job to help her out. Write a function called `findCombination` that takes three `int` inputs, and outputs a `char` representing the correct operation (`+` or `*`), or `N` if no such combination is possible.

Function Name:	`findCombination`

Input(s):	`int x`
	`int y`
	`int z`

Return value:	A `char`: `+` if two of the inputs can be added to make a third, `*` if multiplied, and `N` if there is no combination.
Special requirements:	If there is both a way to add **and** multiply, then it doesn't matter which is returned.
Hints:	The order of the inputs does not matter!

Example usage:	`findCombination(10, 10, 100)` returns `*`
	`findCombination(-5, 0, 5)` returns `+`
	`findCombination(67, 1, 66)` returns `+`
	`findCombination(100, 1, 55)` returns `N`

b) Opposite Day (2 points)

Every once in a while, Marian likes to mix things up and have "opposite day", on which they all talk backwards and never mean what they say. On opposite day, they play "opposite combinations" – rather than testing to see if the numbers add or multiply to one another, they test to see if they **subtract** or **divide** to one another (using regular math division, **not** Java int division).

Marian is even worse at this game. Write a function called `findOppositeCombination` to help her!

Function Name:	`findOppositeCombination`

Input(s):	`int x`
	`int y`
	`int z`

Return value:	a `char`: `-` if two of the inputs can be subtracted to make the third, `/` if two of the inputs can be divided to make the thirds, and `N` if there is no combination
Special requirements:	If there is both a way to subtract **and** divide, then it doesn't matter which is returned.
Hints:	You should call the function `findCombination`... The order of the inputs does not matter!

Example usage:	`findOppositeCombination(10, 10, 100)` returns `/`
	`findOppositeCombination(-5, 0, 5)` returns `-`
	`findOppositeCombination(67, 1, 66)` returns `-`
	`findOppositeCombination(100, 1, 55)` returns `N`

c) Color combinations (3 points)

Sometimes the kids get bored of playing such a simple game, so Marian spices things up with **colors**! In "color-combintation", the players are given three integers, and need to determine whether they

add or multiply to one another (this is like refular "combinations"). The additional catch is that she will also reveal a ball, which will either be **red**, **blue**, or **both**. If the ball is red, the first and third integers are multiplied by 3. If the ball is blue, then the second integer is multiplied by 2. If the ball is both red and blue, then both happen.

For example, suppose the numbers are `1`, `27`, and `3`. Suppose `isRed` is true and `isBlue` is false. Since `isRed` is true, we multiply the first and third inputs by 3 (now they are `3` and `9`. Since `isBlue` is false, we don't do anything to the second input. Then the numbers we test are `3`, `27` and `9`. We would return `'*'`, since `3*9=27`. If instead `isBlue` were true, then the second number would be `27*2=54`. Then the numbers would be `3`, `54` and `9`, which can't combine into one another, so we would return `'N'`.

Marian is **even worse** at this game. Write a function called `findColorCombination` to help her!

Function Name:	`findColorCombination`

Input(s):	`int x`
	`int y`
	`int z`
	`boolean isRed`
	`boolean isBlue`

Return value:	A `char` that is either `+`, `*`, or `N`, according to the rules in the paragraph above...
Special requirements:	None
Hints:	You can use the function you wrote in part (a)...

Example usage:	`findColorCombination(5, 5, 10, false, false)` returns `+`
	`findColorCombination(5, 2, 10, true, false)` returns `*`
	`findColorCombination(5, 2, 10, true, true)` returns `N`
	`findColorCombination(5, 2, 9, false, true)` returns `+`

Part ∞: Feedback Form [3 points]

Fill out this feedback form **after you submit** this assignment. Completion of this will count towards your grade, but your responses themselves will not affect your grade in any way (so be honest!).

What to turn in

On Canvas, turn in a zip folder named `<your_net_id>_P3.zip` containing the files:

- `AgeRestriction.java` [3 points]
  - `int isOver13(int age)` [1 point]
  - `void whatCanIBorrow(int age)` [2 point]
- `OrganizeBookshelf.java` [3 points]
  - `int getRightmostBookID(int id_1, int id_2)` [1 point]
  - `int spaceToLeave(int id_1, int id_2)` [2 point]
- `LateFees.java` [3 point]
  - `boolean isAccountBad(boolean paidLateFee, boolean hasOverdueBook)` [2 point]
  - `boolean isAccountGood(boolean paidLateFee, boolean hasOverdueBook)` [1 point]
- `Combinations.java` [8 point]
  - `char findCombination(int x, int y, int z)` [3 points]
  - `char findOppositeCombination(int x, int y, int z)` [2 point]
  - `char findColorCombination(int x, int y, int z, boolean isRed, boolean isBlue)` [3 point]

Also complete the feedback form. [3 points]
